



# 基于软件体系结构开发的案例



# 票务系统架构设计案例分析

- 1 项目背景
- 2 需求分析
- 3 系统架构设计
- 4 小结

# 1. 项目背景

由于票务种类的繁多，客户信息的量大复杂。所以在其管理上存在较大困难，特别是早期单用人力和纸张进行管理。导致信息的不全面和错误率高，加之存储介质的约束，难以长期有效的管理。

随着计算机网络的发展，电子商务的普及。一种基于B/S模式的票务系统提出了需求。由于票务的特殊性，需要系统有很强的稳定性，要求较快的反应速度，响应多点同时请求。另外后台对票务的所有相关信息需要完全记录。完成历史信息的保存，查询；对当前信息的录入，查询，修改，删除。



## 2. 需求分析

### 主要任务

创建代表“目前”业务情况的业务模型，并将此业务模型转换成“将来”的系统模型，包括功能需求和非功能需求。非功能需求又包括质量属性和各种约定。

通过对客户的当前业务的分析，我们得到当前业务的基本需求。

# 功能需求

功能	说明
客户信息管理	用户的创建、登录、删除和维护
票务信息管理	票务的添加、删除和维护
票务查询	查看相应的票务信息
预定购票	票务的预定、购买和取消

## 非功能需求

质量属性	说明
性能	用户访问的系统应该能在规定的时间内做出响应，如果系统由于网络或者数据库原因不能在规定时间内做出反应，那么系统应该提出警告，不能出现用户无故长时间等待的情况。
安全性	在web数据库客户端，web服务器和数据库服务器之间，都应该有防火墙保护，防止网络上的非法数据请求。
易用性	不同的用户应该能够以不同形式访问不同的内容
可用性	系统提供7X24小时的服务，且很少停机
可测试性	系统是的各部分易于单独测试，并能方便地进行整体测试

# 质量场景分析

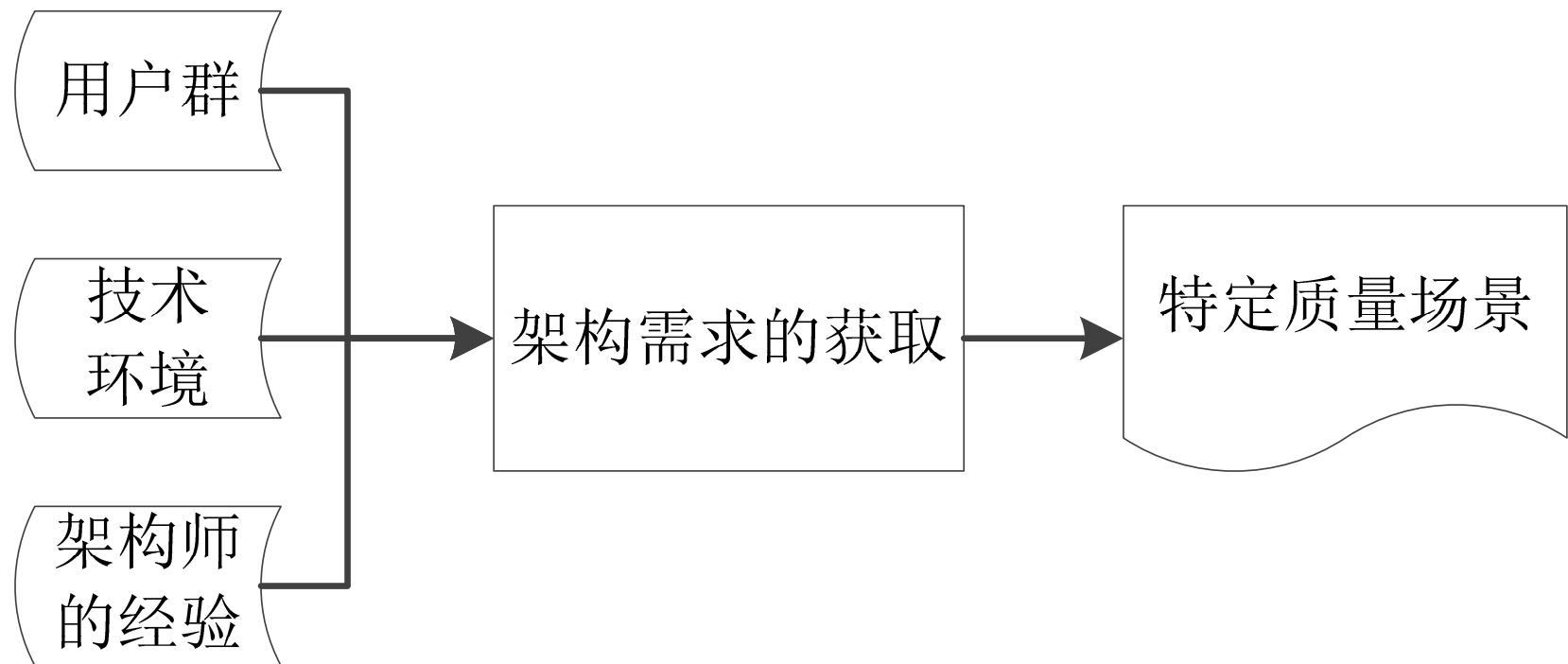
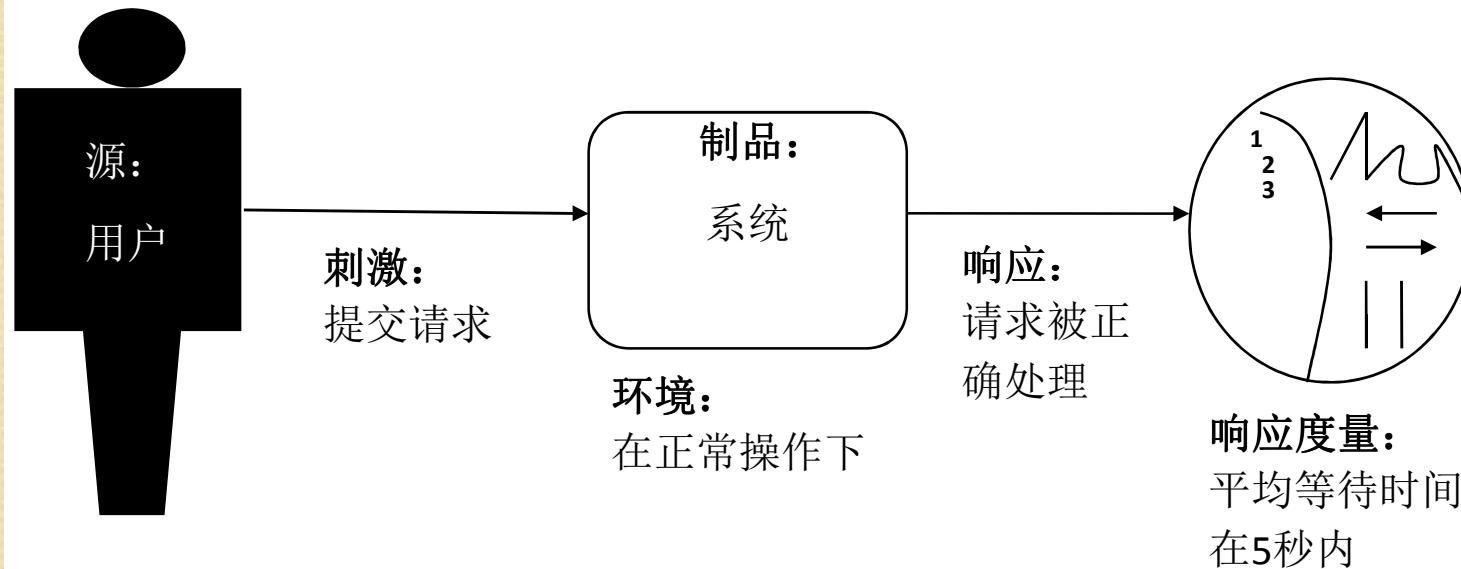


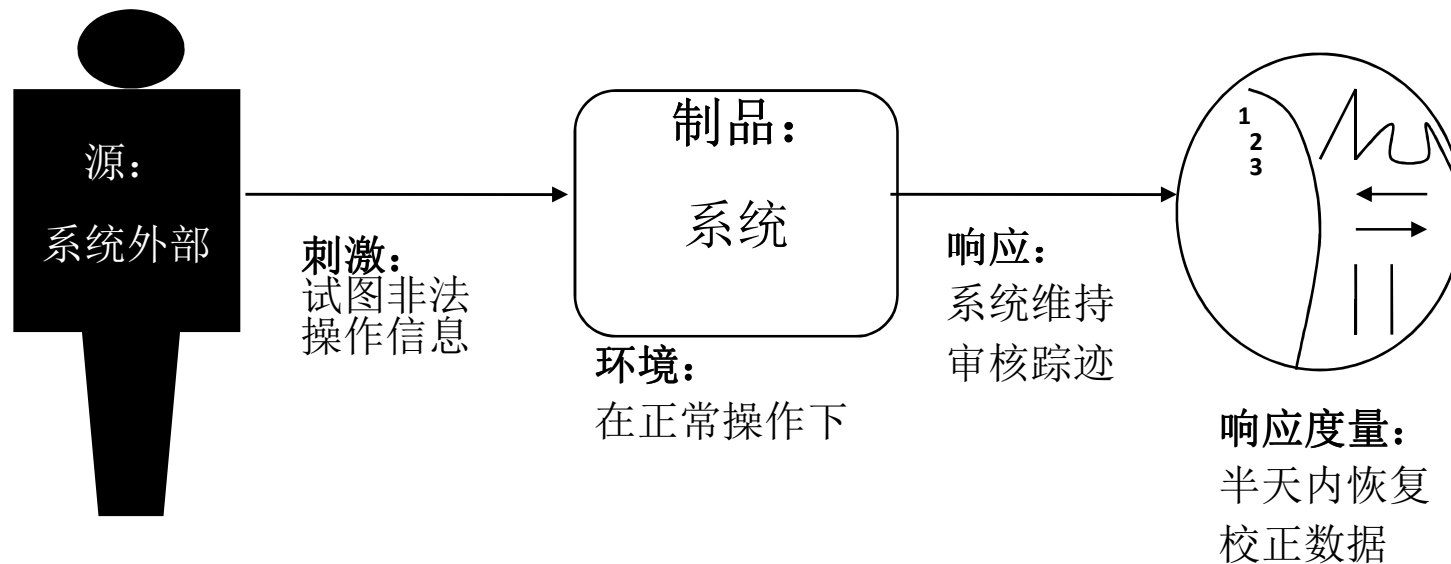
图7.2 架构需求的获取阶段任务描述图

- 1) 性能场景：在系统处于高峰时期，保证登陆的每个顾客所作的选择和查询的响应时间能在5s以内，如果需要等待则给出有友好的提示。系统可以保证以最快速度同时响应500个用户的操作。

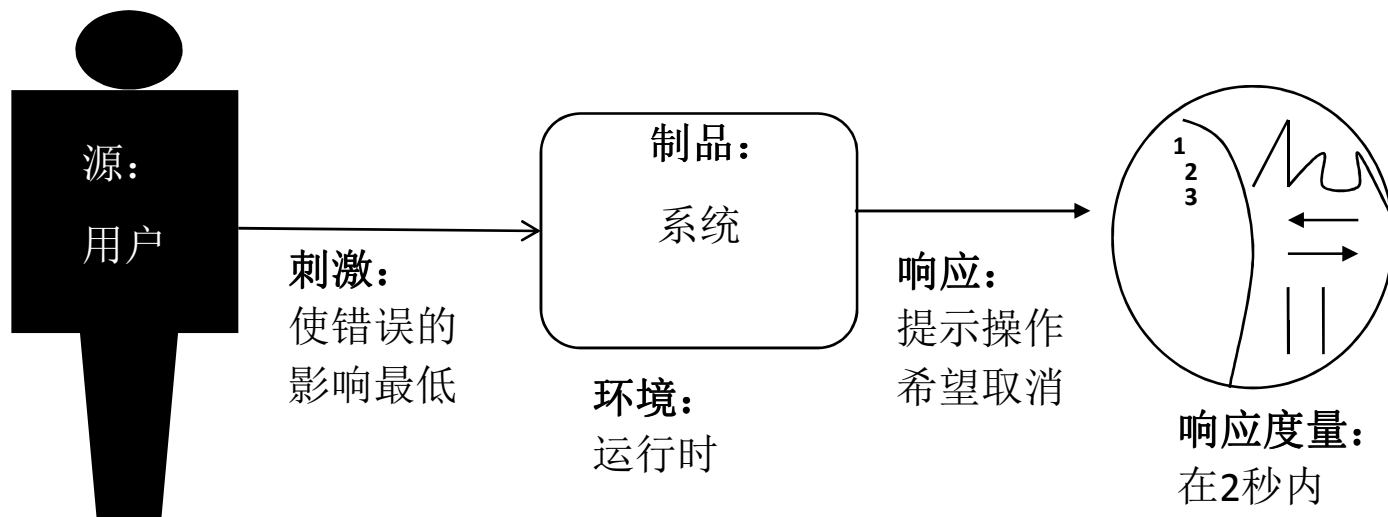




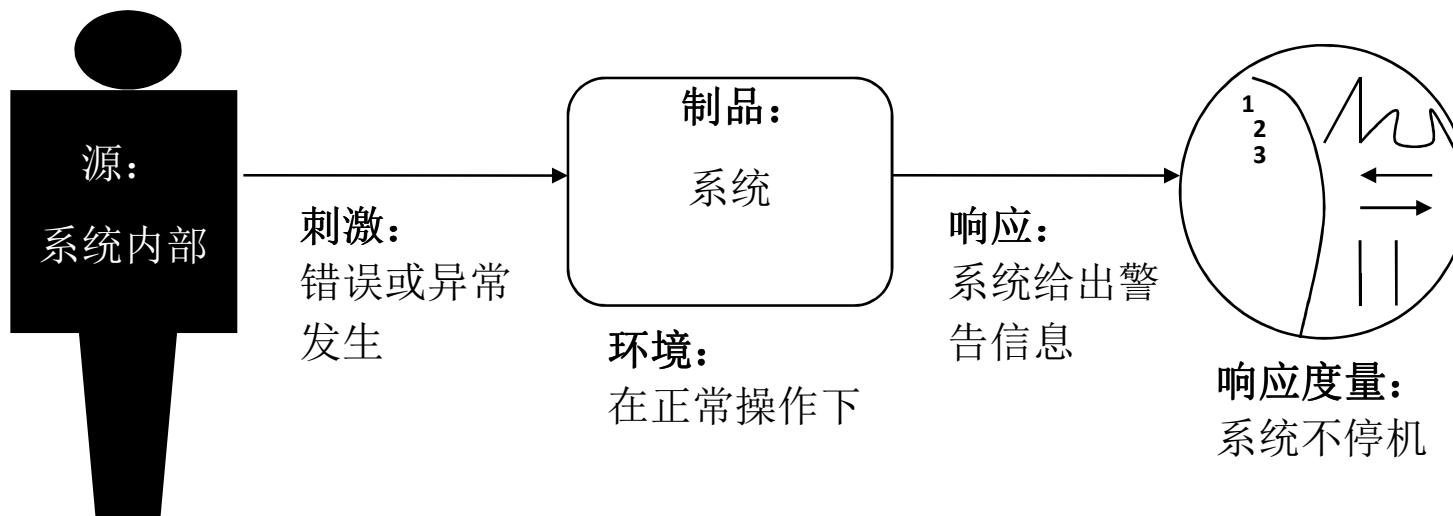
- 2) 安全性场景：杜绝非法用户试图绕过应用服务器直接连接到数据库服务器的端口上，防止非法窃取注册用户个人信息；屏蔽某IP短时间内的海量无意义的访问，以防被挤爆，使正常用户无法使用。保证系统数据的机密性和完整性。



3) 易用性场景：在该系统中，用户希望在运行时能尽快取消某操作使错误的影响降到最低，取消在1秒内发生；要求具有基本电脑操作常识的人，可以根据良好的界面设计迅速学会使用方法，让熟手用户使用快捷键。



4) 可用性场景：在正常的工作时间内，系统必须具有极高的可用性，保证出故障几率最低。出现故障时系统有相应的处理机制。





## 功能场景分析

根据业务的功能需求，该系统主要的涉众有系统管理人员和客户，系统管理人员又分为票务管理人员和用户管理人员。票务管理人员会对票务信息进行相关维护，用户管理人员对客户信息进行相关的维护。由此得出系统角色，分析其对系统的具体要求，并找出系统的各个用例。

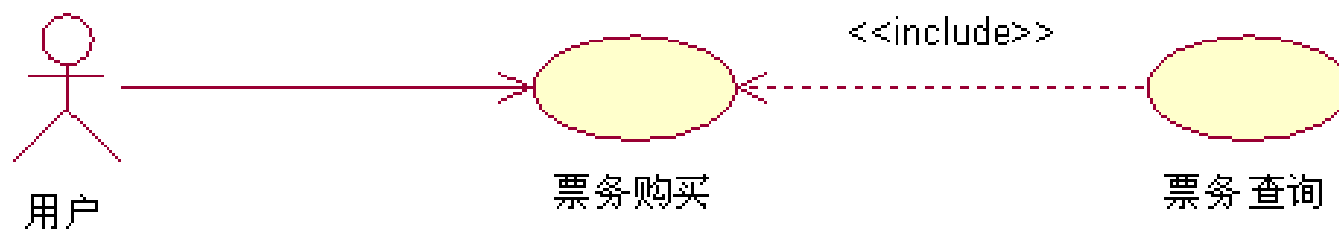
用例	说明
票务信息查询	用户输入相关查询条件信息，查看到相关票务的具体信息，当查询条件不符合规定时，系统给出相应提示。
票务操作	用户根据查询出来的票务信息对票务信息进行预订，购买，取消等操作
票务信息维护	票务管理员对票务信息进行维护，如添加，删除等
用户信息维护	用户管理员根据用户资料，维护系统中记录的用户相关信息。
... ..	... ..



## 2.2 细化定义

### (1) 细化用例

细化业务用例模型，是为了更加详细地分析和描述用例。同时，将业务用例模型转换成系统的用例模型。下面，以“用户”角色进行票务购买为例。



细化用例后，还需对用例进行详细描述，直到所有涉众都认可描述的内容已经能够正确表达出他们的需求为止。在RUP方法论中指明通过阐述一个用例的名称、简要描述、事件流、特殊需求、前置条件和后置条件等六个方面可以对用例进行描述。下面以用例“用户购买票务”为例细化描述。

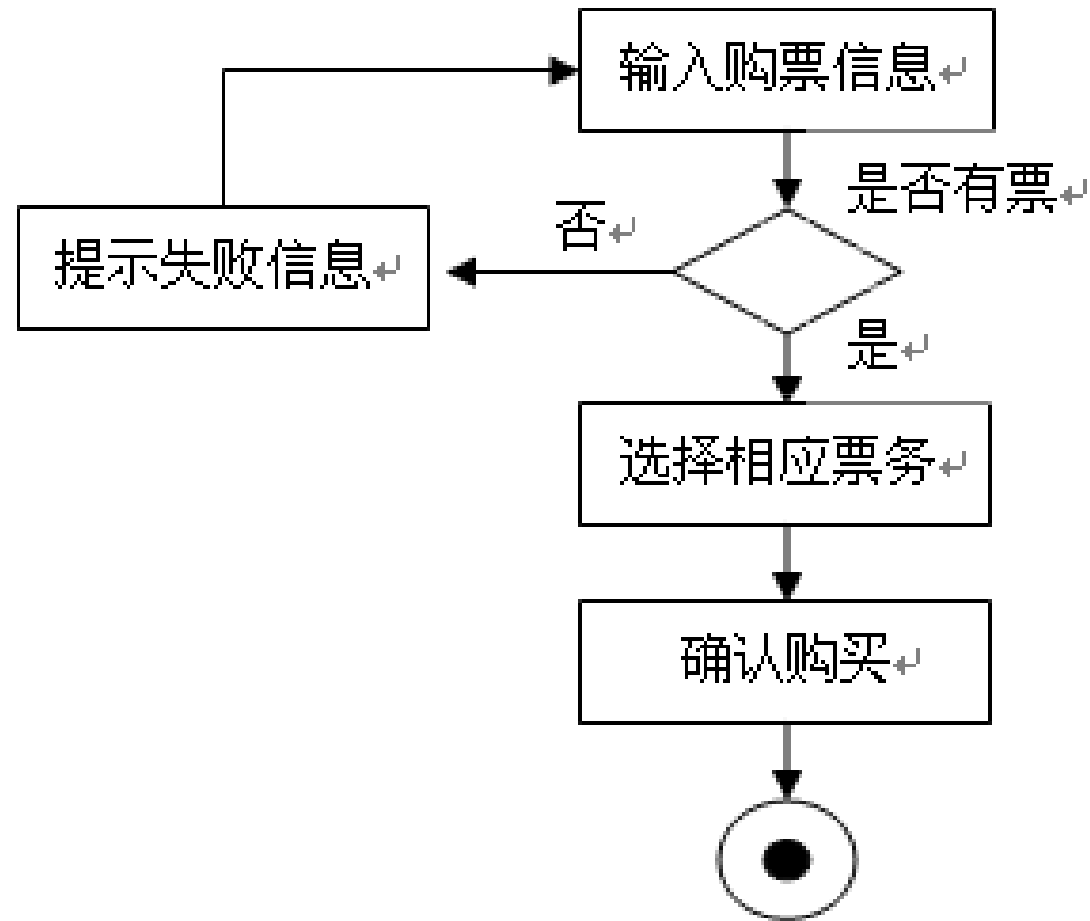
要素	说明
用例名称	用户购买票务
简要描述	用户根据当前票务信息购买相应票务
事件流	<p>基本事件流</p> <ul style="list-style-type: none"><li>(1) 用户在购票的名称栏中输入要购买的票务的起始地与目的地</li><li>(2) 系统根据客户输入，列出相应的票务信息</li><li>(3) 用户根据自己的实际情况选择符合自己相应条件的票务，如票价、时间等。</li><li>(4) 系统显示购买成功，或者显示交易失败。</li><li>(5) 该“用户购买票务”用例结束。</li></ul>



## “用户购买票务”用例细化描述（续）

要素	说明
备选事件流	系统查询不到票务相关信息，则按下一步步骤进行： （1）提示用户票务交易无法进行，并给出交易失败原因 （2）其次，撤销此次交易的记录。
特殊需求	系统不可伪造数据，交易失败原因要合理并且详尽
前置条件	用户必须先登陆
后置条件	交易成功后数据库及时更新票务信息

上面对用例的描述仅限于文字描述，还不够形象。再以活动图的形式进行建模描述如下：



## (2) 结构化用例

结构化用例的目的是通过观察这些已经细化的用例，看能不能抽取出共有的、可选的行为，把这些共同的内容建立为新的用例。这样的好处是，可以消除冗余的需要以及改善系统整体需求内容的可维护性。像“用户信息维护”用例中，“查询用户信息”应作为一个新的用例提取出来，以提高上面所说的需求内容的可维护性。

### 3. 架构设计

将需求内容转换成设计模型的雏形以及用户体验模型，其目的是建立整个系统初步的解决方案，为详细设计活动打下基础。

其步骤如下：

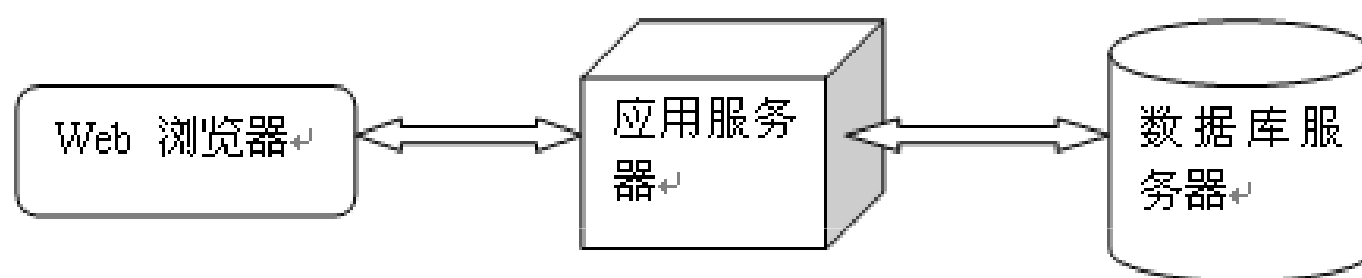
- (1) 基本架构设计
- (2) 架构文档化
- (3) 架构评估


### 3.1 基本架构设计

早期的票务系统仅仅针对售票单位，只是简单的数量控制，票务记录。而新的票务系统不仅仅具有以前的所有功能，还利用网络将客户包括进来。方便客户进行操作，利用网络可以让客户在任何有网络的地方就可以直接连入系统。又由于计算机的支持，数据库中有所有客户的信息，可以方便售票方对客户进行管理，提供更好的服务。

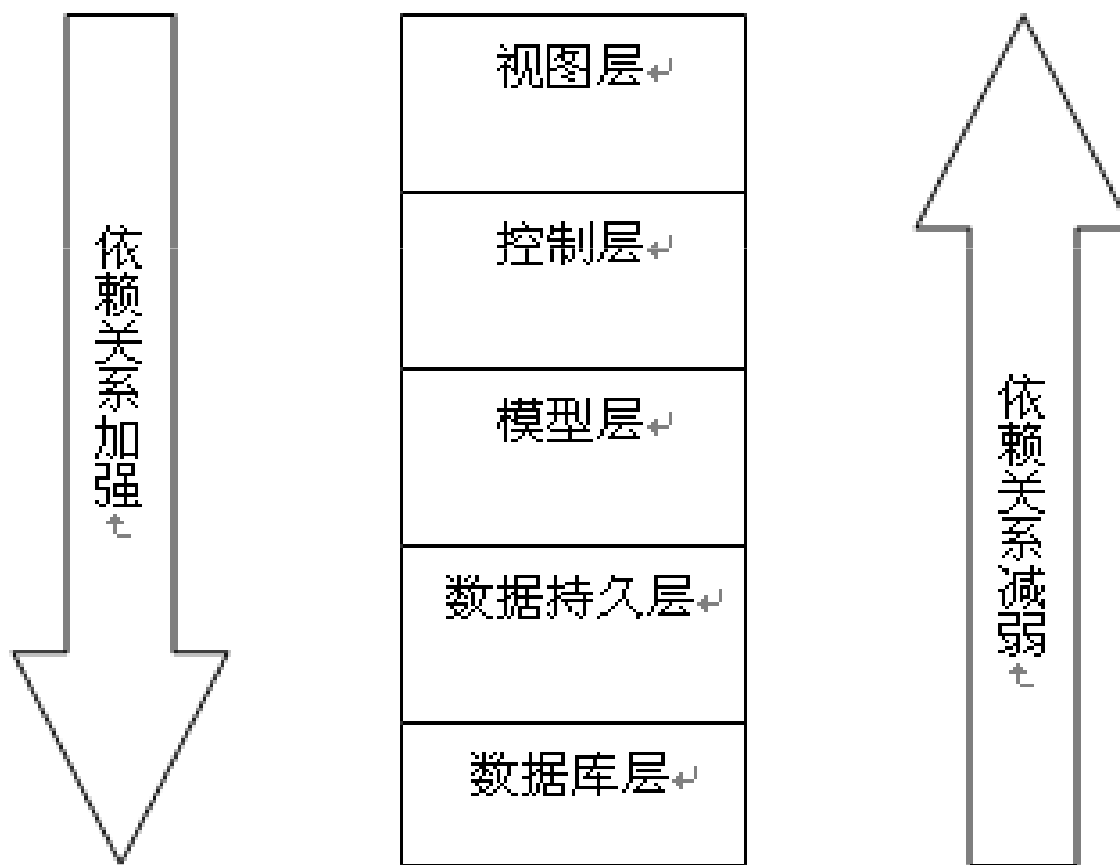
本系统采用基于B/S的分层结构。这种结构有如下特点：节省投资、跨地域广；维护和升级方式简单，如果想对功能修改，可以方便的进行更改，大大减少维护成本。

系统的结构视图如下：




- 
- 在J2EE开发中，搭配良好的框架可以降低开发人员解决复杂问题的难度，而如何将框架整合起来，以使每一层都向另外的层次以松散的方式来提供接口，同时让组合的三个框架在每一层都以一种松耦合的方式彼此沟通，从而与低层的技术透明无关，这就是框架分析的目的和要求。
  - 根据前期对需求的分析，决定采用基于SSH框架来构建此分布式的信息管理系统。
  - SSH是 struts+spring+hibernate的一个集成框架，是一种Web应用程序开源框架。
  - SSH把Struts、Hibernate和Spring组合起来，目标就是希望能实现系统的“低耦合、高内聚”。也就是要求系统易于维护、易于适应变更、可重用性的特点。

SSH多层的构架模式，从上到下依次为视图层、控制器层、模型层、持久化层和数据库层，如下图所示：







视图层：职责是提供控制器，将页面的请求委派给其他层进行处理，为显示提供业务数据模型。

控制层：职责是按预定的业务逻辑处理视图层提交的请求。（1）处理业务逻辑和业务校验

（2）事务管理

（3）管理业务层对象间的依赖关系

（4）向表示层提供具体业务服务的实现类

模型层：职责是将模型的状态转交视图层或者控制层，以提供页面给浏览器。

数据持久层：职责是建立持久化类及其属性与数据库中表及其字段的对应关系。提供简化SQL语句的机制。实现基本的数据操作（增、删、读、改）

数据库层：数据库的建立与管理。



规则（约束）：

- （1）系统各层次及层内部子层次之间都不得跨层调用。
- （2）由bean传递模型状态。
- （3）需要在表示层绑定到列表的数据采用基于关系的数据集传递。
- （4）对于每一个数据库表（Table）都有一个DB Entity class与之对应，由Hibernate完成映射。
- （5）有些跨数据库或跨表的操作（如复杂的联合查询）也需要由Hibernate来提供支持。
- （6）表示层和控制层禁止出现任何SQL语句。

## SSH框架介绍:

视图层、控制器层用Struts框架来实现，模型层的功能Spring来完成。持久化层时使用Hibernate实现，在这层使用了DAO模式。

Struts应用MVC模型使页面展现与业务逻辑分离，做到了页面展现与业务逻辑的低耦合。当充当表示层的页面需要变更时，只需要修改该具体的页面，不影响业务逻辑Bean等；同样，业务逻辑需要变更的时候，只需要修改相应的Java程序。

使用Spring能运用IoC技术来降低了业务逻辑中各个类的相互依赖：假如，类A因为需要功能F而调用类B，在通常的情况下类A需要引用类B，因而类A就依赖于类B了，也就是说当类B不存在的时候类A就无法使用了。使用了IoC，类A调用的仅仅是实现了功能F的接口的某个类，这个类可能是类B，也可能是类C，由Spring的XML配置文件来决定。这样，类A就不再依赖与类B，耦合度降低，重用性得以提高。

使用Hibernate能让业务逻辑与数据持久化分离，就是将数据存储到数据库的操作分离。在业务逻辑中只需要将数据放到值对象中，然后交给Hibernate，或者从Hibernate那里得到值对象。至于用DB2、Oracle、MySQL还是SQL Server，如何执行的操作，与具体的系统无关，只需在Hibernate的相关的XML文件里根据具体系统配置好。

# 质量属性的满足

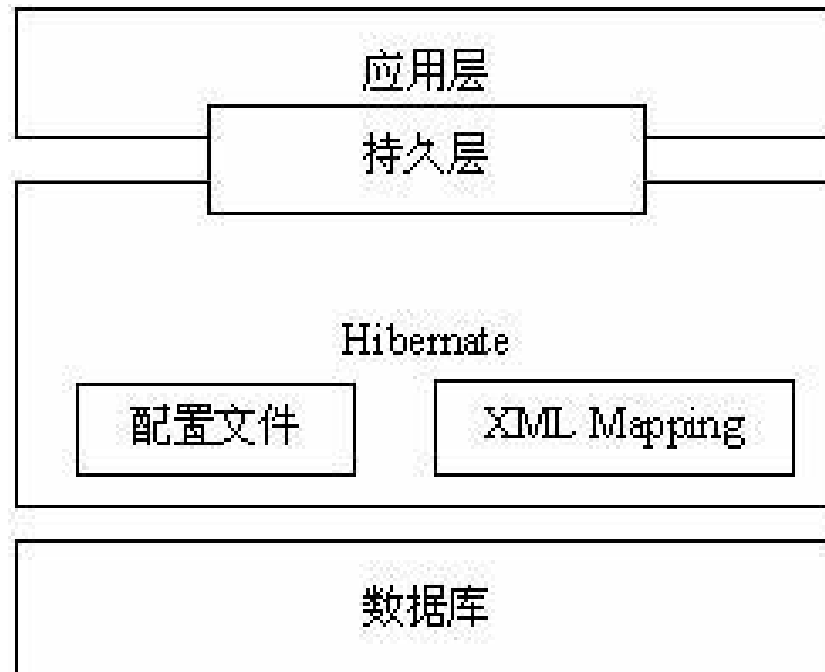
现在我们通过下表来看看构架是如何来满足系统的关键质量属性需求。

目标	实现方式	所采用的战术
性能	用户访问的系统应该能在规定的时间内做出响应，如果系统由于网络或者数据库原因不能在规定时间内做出反应，那么系统应该提出警告，不能出现用户无故长时间等待的情况。	限制队列大小 缓冲池技术
易用性	遵从 J2EE 的系统提供了诸如 JSP 和 servlet 这样的 Java 技术，它们支持内容的渲染，以满足不同用户的需要。用户对系统的操作能得到正确及时的反馈。	单独的用户接口 支持用户主动
安全性	遵从 J2EE 的系统提供了由容器进行授权校验的基于角色的安全性机制，以及已经为使用做好准备的在程序中进行授权检查的安全性机制，在多个用户进行并发操作时保持数据的一致性，有效性。	身份验证 授权 数据机密性 验证码 维护完整性
可用性	当系统试图超出限制范围来进行票务查询或者订购票时必须进行错误检测并且抛出异常，中止进一步的错误操作。遵从 J2EE 的系统提供了可以使用的事务服务，通过提供内建的故障恢复机制，提高了应用的可用性和可靠性	异常检测 内建故障恢复机制 资源调度策略

## 数据持久层的架构分析

在数据持久层，我们使用Hibernate来进行处理，通过下面我们来看看如何通过Hibernate来满足系统的质量属性需求。

Hibernate体系结构概要图



# 数据持久层的架构分析

## Hibernate满足的质量属性需求

目标✎	实现方式✎	所采用的办法✎
性能✎	当应用程序需要在关联关系间进行导航的时候，由 <b>Hibernate</b> 获取关联对象。同时， <b>Hibernate</b> 的 <b>Session</b> 在事务级别进行持久化数据的缓存操作。✎	抓取策略✎ 缓存机制✎
安全性✎	并发操作时，保证数据的排他性✎	使用锁机制✎
易用性✎	用户在进行 <b>CRUD</b> 操作请求时，可以得到 <b>Hibernate</b> 的及时处理，迅速得到反馈。✎	封装 <b>JDBC</b> ✎



### (1) 性能

Hibernate本质上是包装了JDBC来进行数据操作的，由于Hibernate在调用JDBC上面是优化了JDBC调用，并且尽可能的使用最优化的，最高效的JDBC调用，所以性能令人满意，同时应用程序需要在关联关系间进行导航的时候，由Hibernate获取关联对象，Hibernate提供的对持久化数据的缓存机制也对系统的性能的提高起了很大的作用。

### (2) 安全性

Hibernate提供的悲观锁/乐观锁机制，能够在多个用户进行并发操作时保持数据库中数据的一致性与完整性，避免了对数据库中数据的破坏。

### (3) 易用性

用户在对票务信息进行操作时都得到Hibernate的支持。

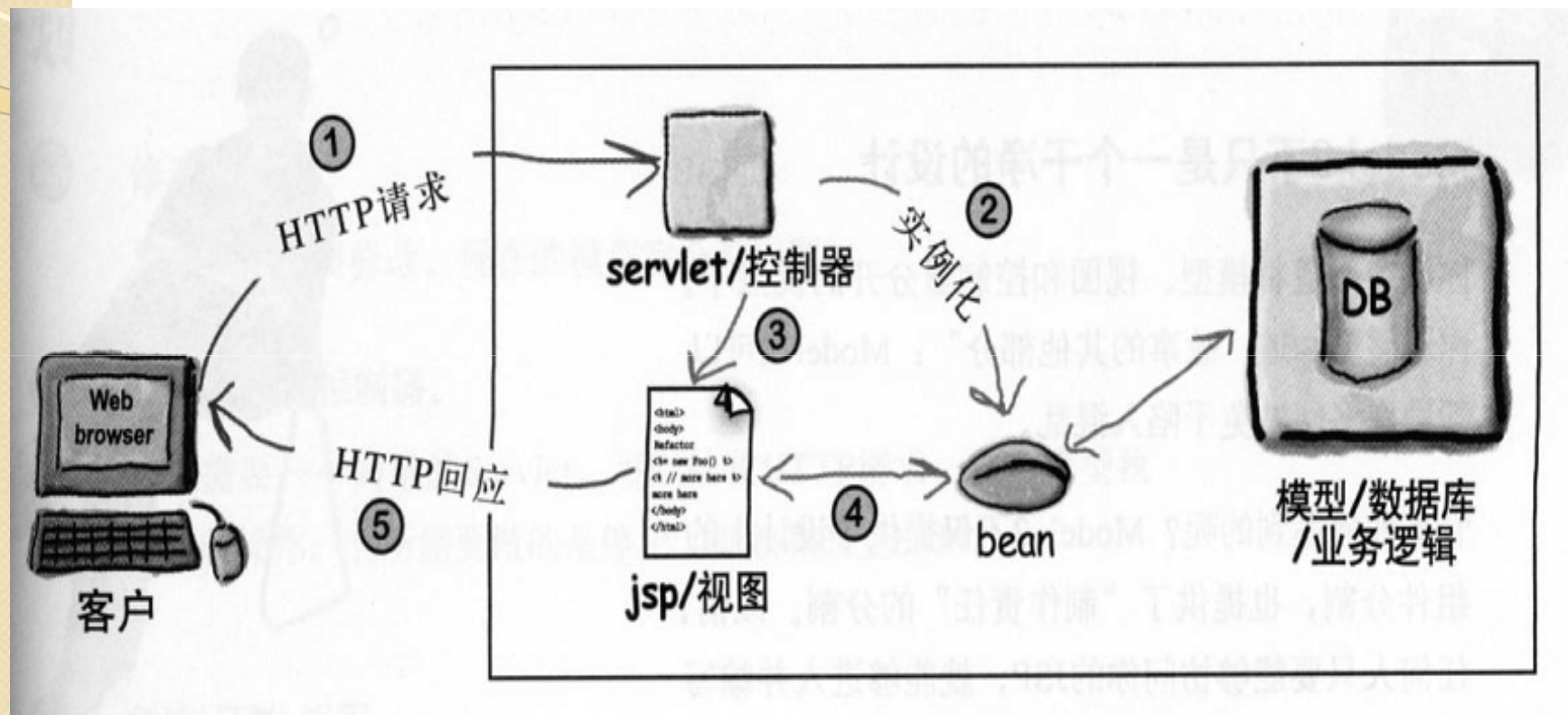
# 业务逻辑层的架构分析

业务逻辑层作为该系统的关键部分，对系统的灵活性实现起着决定性的作用。在本系统的业务逻辑层架构层中，采取了MVC模式，下面简单介绍一下MVC模式的好处：

- (1) 实现了客户端表示层和业务逻辑层的完全分离
- (2) 高效可靠的事务处理
- (3) 具有良好的易用性，安全性



## MVC模式访问流程：

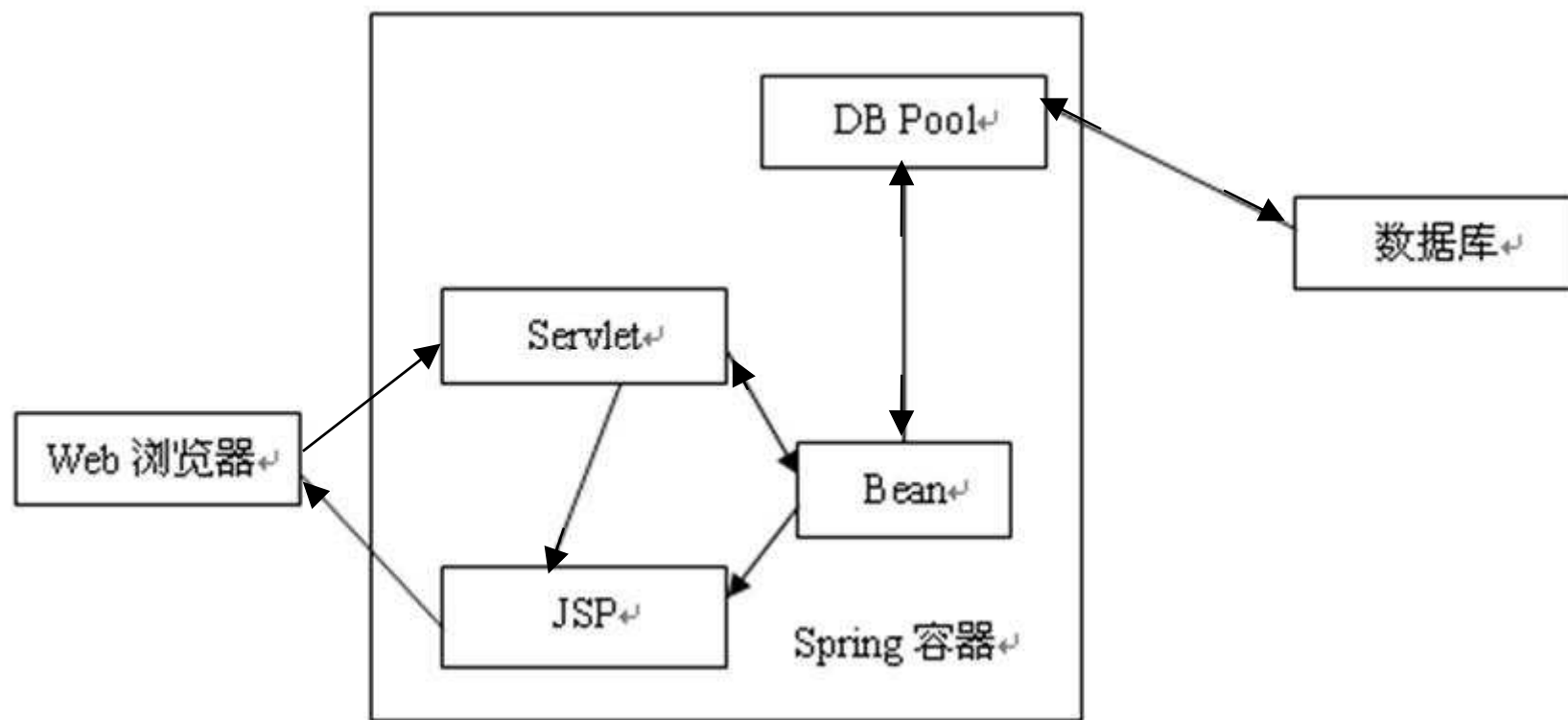



## MVC模式在本系统中应用：

当客户利用网页浏览器，发出HTTP请求时，这通常会牵涉到送出表单数据，例如用户名和密码。Servlet收到这样的数据并解析数据。Servlet扮演控制器的角色，处理你的请求，通常会向模型（一般是数据库）发出请求。处理结果往往以JavaBean的形式打包。视图就是JSP，而JSP唯一的工作就是产生页面，表现模型的视图以及进一步动作所需要的所有控件。当页面返回浏览器作为视图显示出来，用户提出的进一步请求，也会以同样的方式处理。

由于JSP继承了J2EE良好的易用性和安全性，从而为实现系统的关键质量属性奠定了基础。在MVC模式中，视图不再是经典意义上的模型的观察者。当模型发生改变时，视图的确间接的从控制器收到了相当于通知的东西，控制器可以把bean送给视图，以使得视图取得模型的状态。所以，视图在HTTP响应返回到浏览器时只需要一个状态信息的更新。只有当页面被创建和返回时，创建视图并结合模型状态才有意义。这使得提升系统的性能成为可能。只有当相应的操作被执行，系统才会去获取关联对象，并且视图不会直接模型向注册去接受状态信息，使得系统的安全性得到大大提高。

业务逻辑层的框架：





## 业务逻辑层架构分析：

该业务逻辑层的架构是前面MVC模式的一种变形，继承了MVC模式的优点，同时，具体到我们的架构中，它又实现了表示层与业务层的完全分离。在业务逻辑层我们使用Spring框架作为容器，以便实现业务层与表示层和数据层的松耦合。该业务逻辑层架构具备良好的易用性、安全性和性能。

## spring容器如何满足系统关键质量属性

目标↵	实现方式↵	所采用的办法↵
安全性↵	Spring Framework 利用 AOP 来实现权限拦截，还提供了一个成熟的，简洁清晰的安全框架，通过对 spring bean 的封装机制来实现↵	AOP↵ Acegi 安全框架↵
可用性↵	Spring 框架不再强制开发者在持久层捕捉异常，通常持久层异常被包装成 <code>DataAccessException</code> 异常的子类，将底层数据库异常包装成业务异常，开发者可以自己决定在合适的层处理异常。↵ ↵	异常检测↵ 统一的异常处理机制↵

整体的构架如下：

